

FIG 2

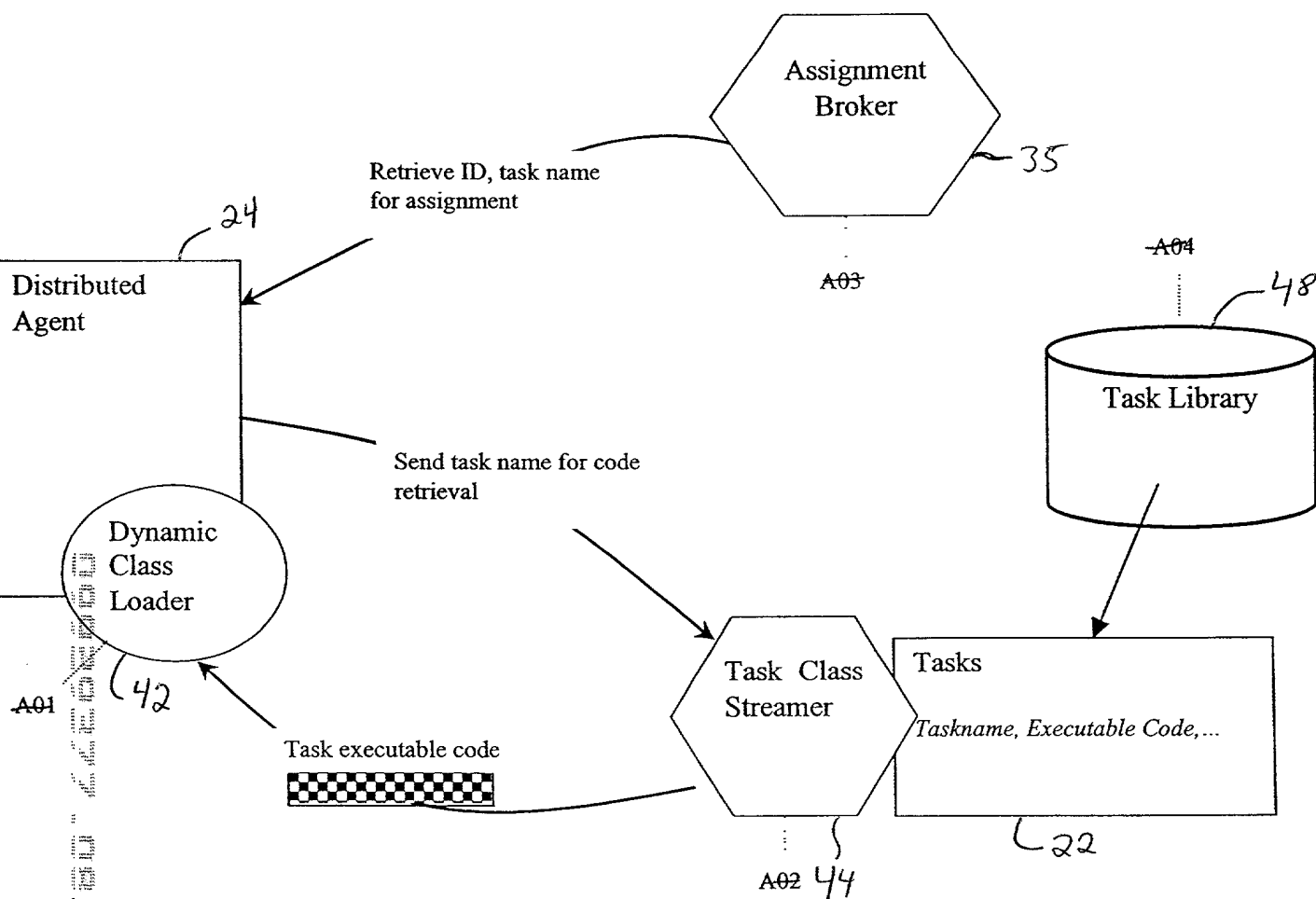


Fig. 3

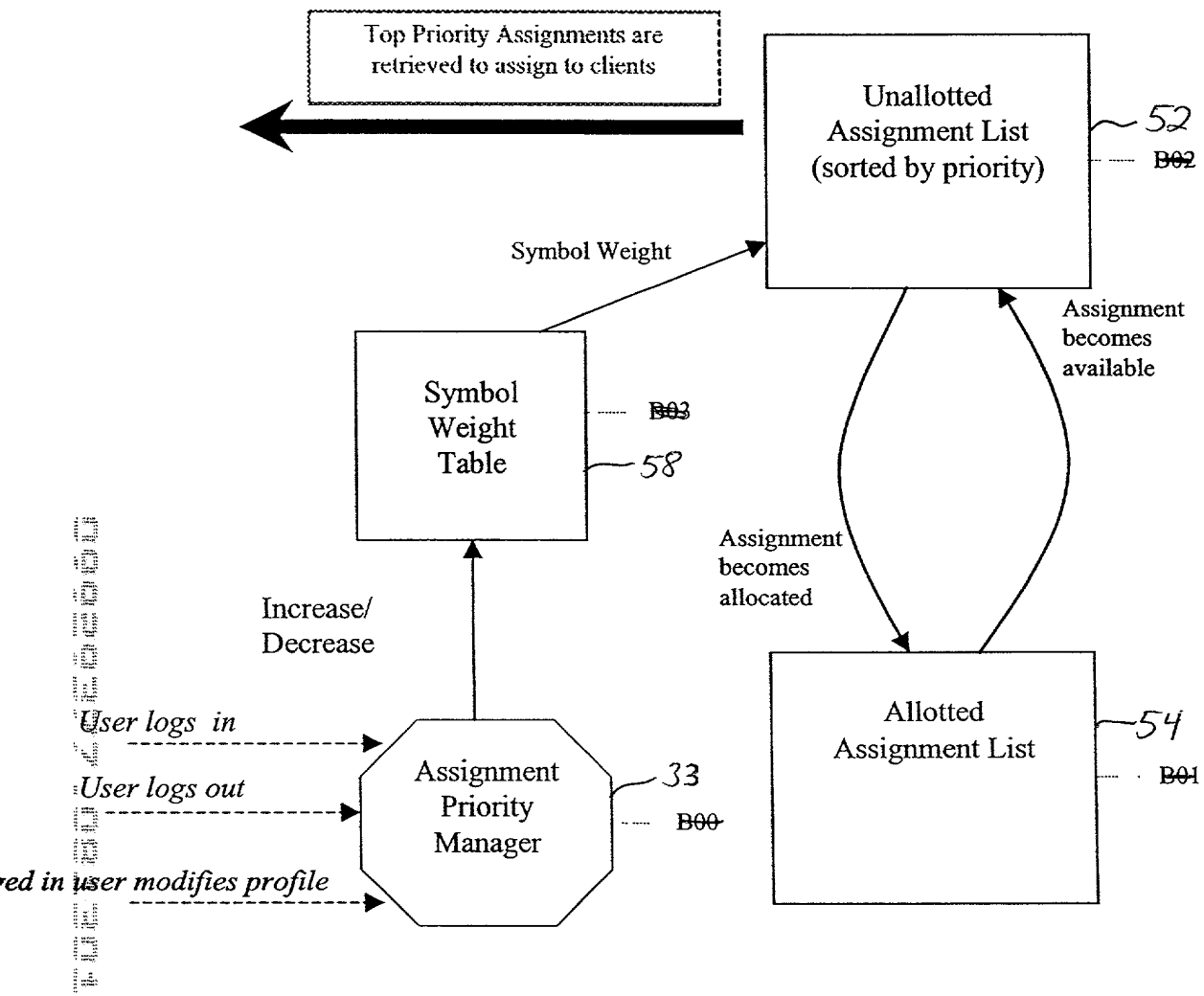


Fig. 4



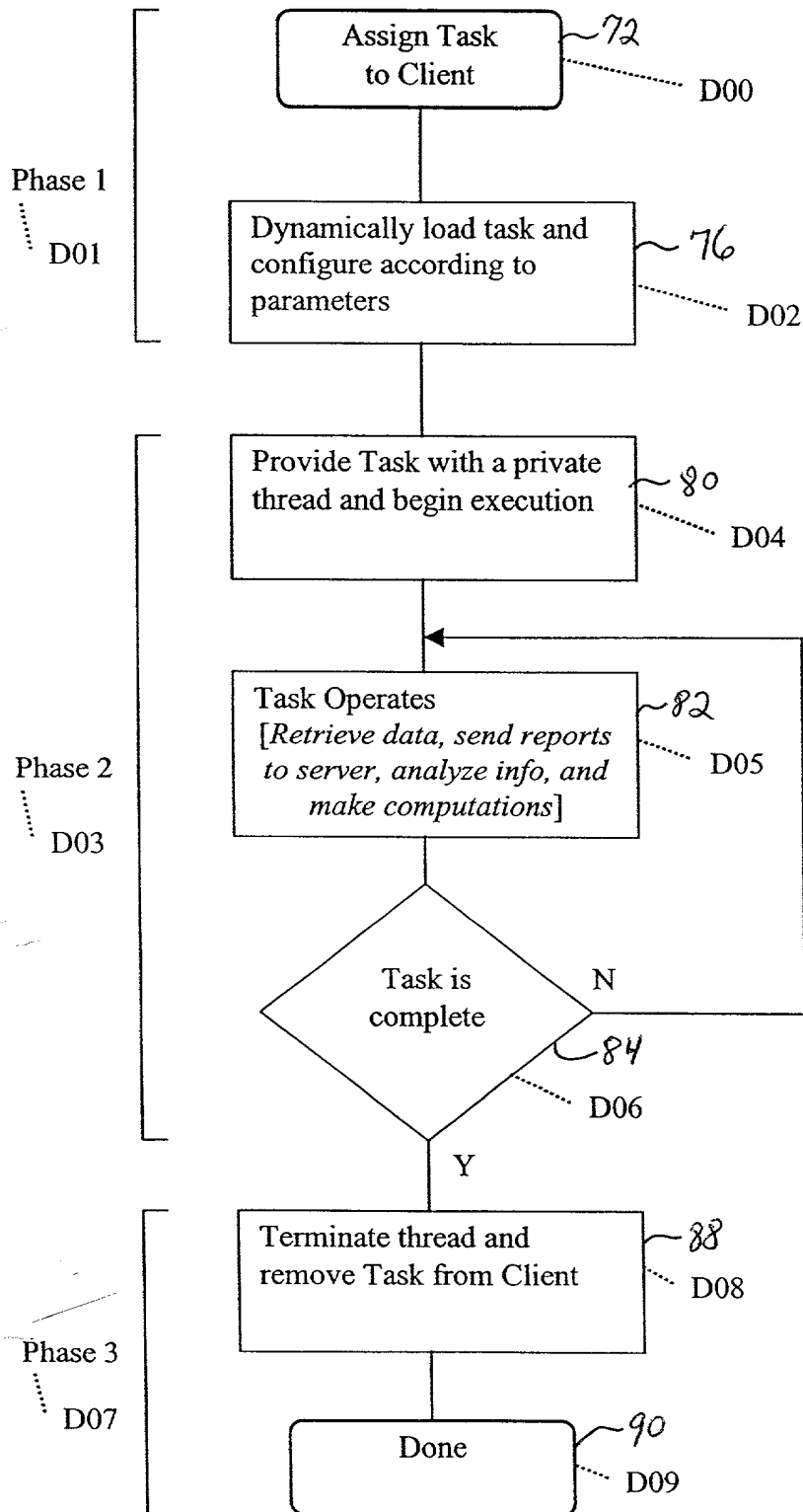


Fig. 6

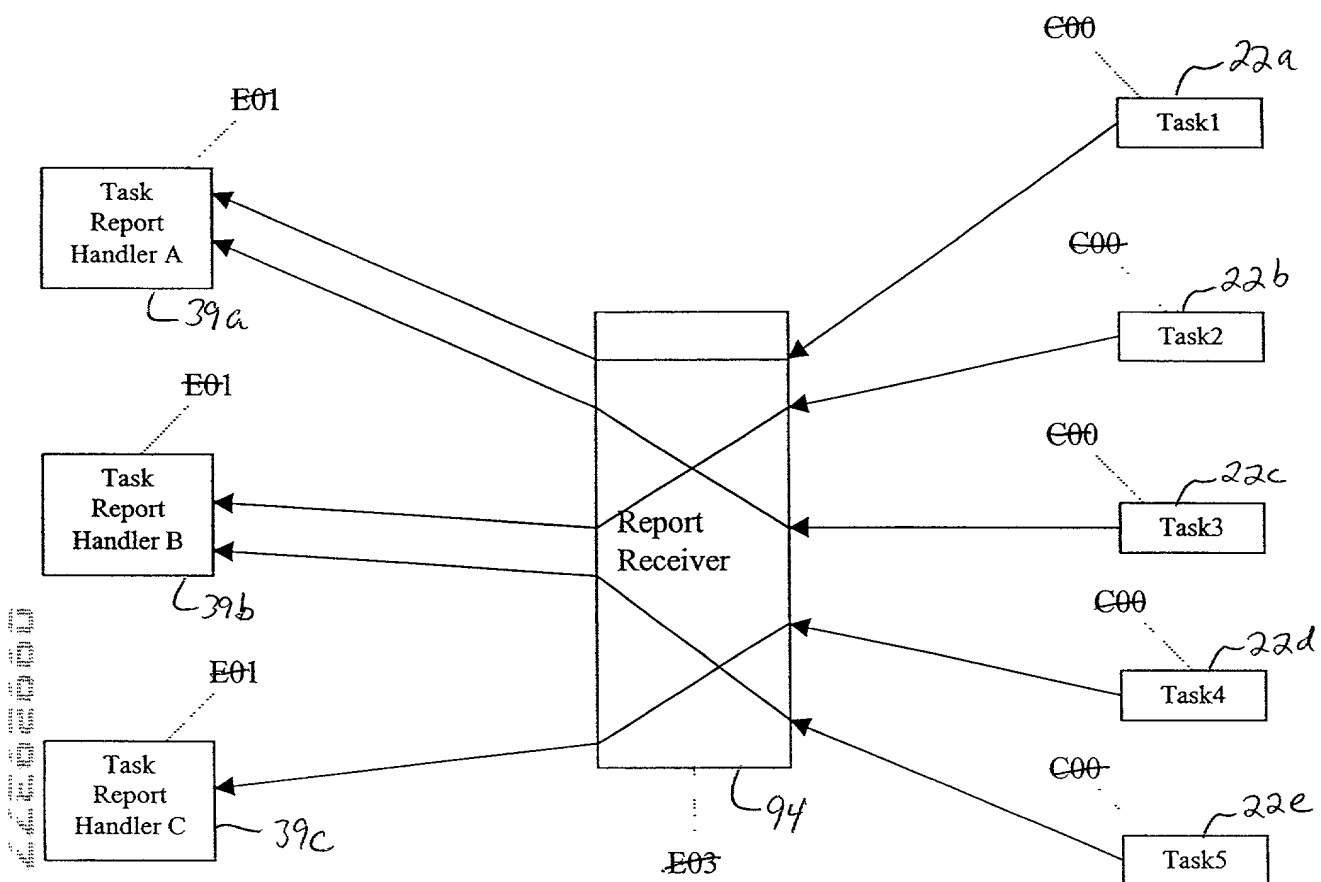


Fig. 7

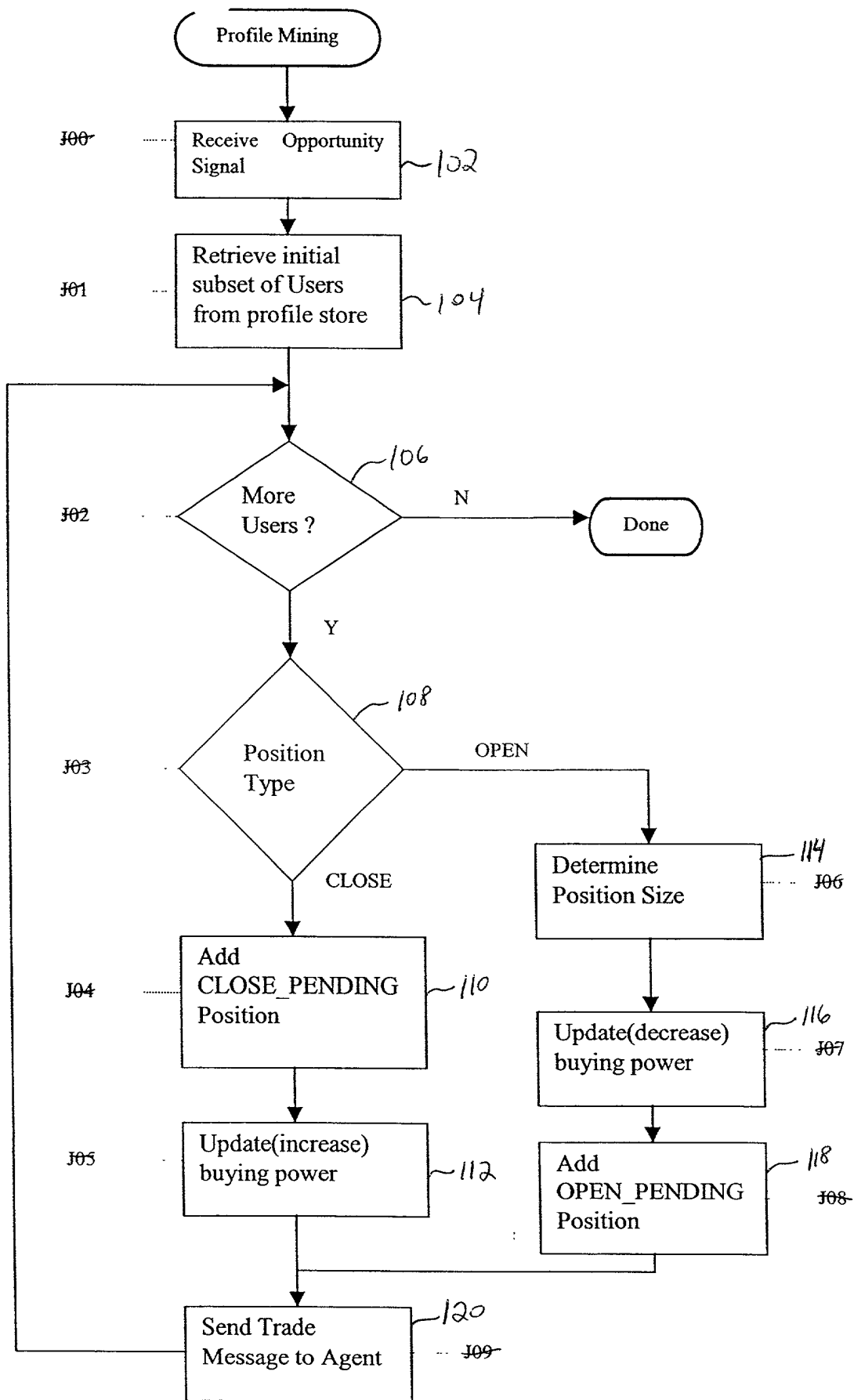


Fig. 8



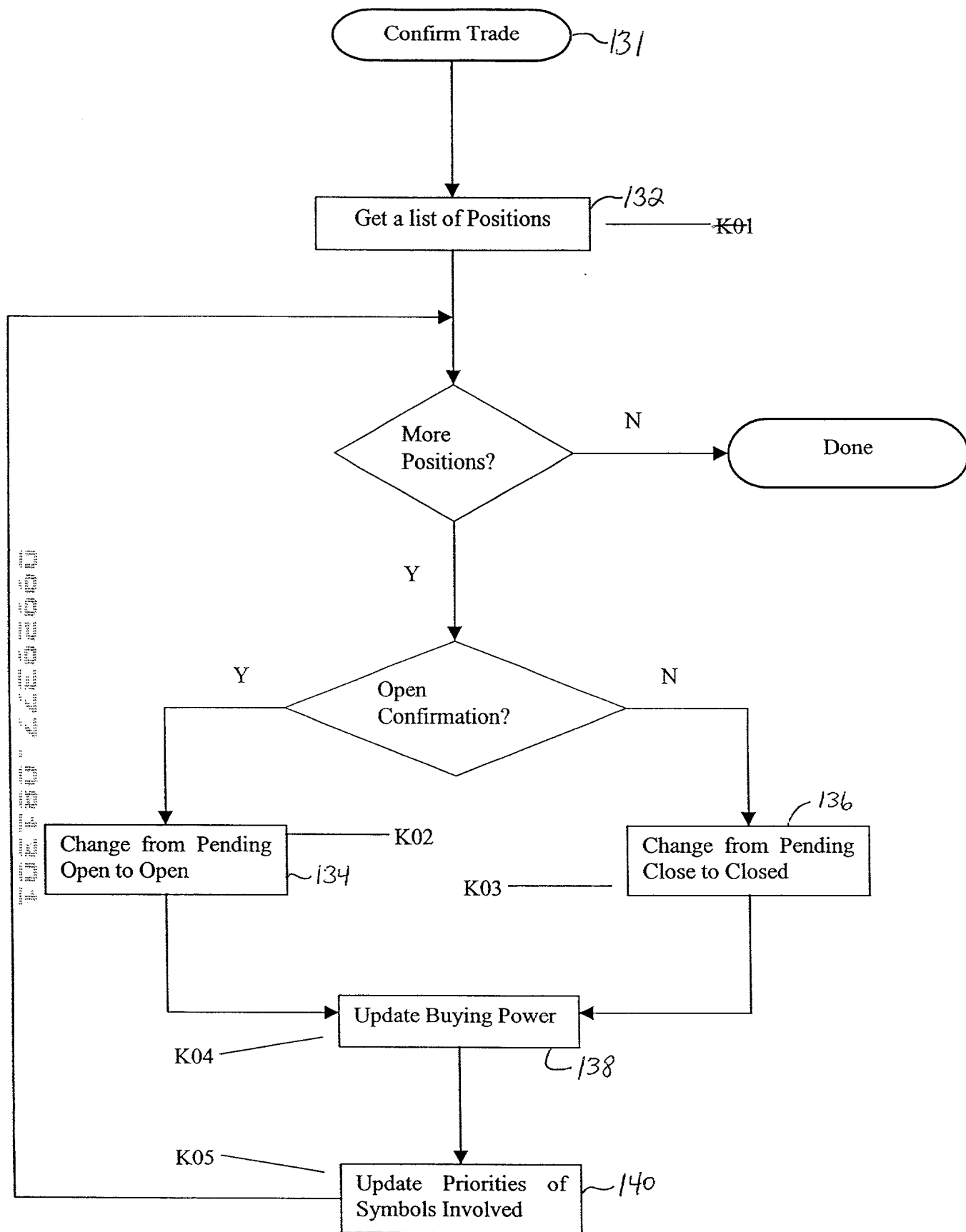


Fig. 9

<u>Trade Message</u>
Action Symbol Quantity Limit Price Trading Module ID Account Information
Confirmation Code Actual Price Timestamp

← 144

Fig. 10

## Investment Profile for <Sample User>

This information is used to customize trading strategies to your needs.

### Trade Parameters

**Amount Per Trade (\$):**

Low end of range	High end of range
Low end of range	High end of range
Low end of range	High end of range

**Shares Per Trade:**

**Price Per Share (\$):**

### Equity Parameters

<b>Volatility</b>	Lowest	Low	Medium	High	Highest	All
<b>Company Size</b>	Smallest	Small	Medium	Large	Largest	All
<b>Volume</b>	Lowest	Low	Medium	High	Highest	All
<b>Preferred Indices</b>	DJIA	S&P500	NASDAQ100			All

SAVE

CANCEL

Fig. 11

```

/**
 * Wrapper class to pass arguments to a task instance on the client.
 */
public class TaskArguments implements java.io.Serializable {

    /**
     * Unique name designating which class this task corresponds to.
     */
    public String taskName;

    /**
     * Execution parameters passed to a task when it is instantiated.
     * This usually takes the form of a Hash Table of objects. The structure
     * is flexible to allow different numbers and sizes of
     * parameter to be passed to particular tasks.
     */
    public byte[] argByteArray;
}

```

Fig. 12A

```

/**
 * A collection of services that a task can utilize during its execution on the
 * client. In order to maintain a high level of modularity, task communication
 * with either the client or server must occur through the methods of this
 * interface.
 */
public interface TaskServiceProvider extends Serializable {

    /**
     * Transmits a report to the server on the wrapped task's request.
     *
     * @param reportText the report to be sent
     */
    public void issueReport( String reportText );

    /**
     * Transmits a request for points to the server on the wrapped task's
     request.
     */
    public void requestPoints();

    /**
     * Creates and installs a NewsDocReceiver for this task with the specified
     feed.
     *
     * @param task the concerned task
     * @param feedKey describes the feed to use
     */
    public NewsDocReceiver installNewsDocReceiver( String feedKey );

    ////////////////////////////////////////////////////
    //Following are service request that tasks need during execution
    ////////////////////////////////////////////////////

    public Vector getQuotes(Vector symbols) throws SB_Exception;

    public void linkToDataFeed(Observer o, Vector symbols);
    public void unLinkFromDataFeed(Observer o, Vector symbols);

    public Vector getNASDAQTopVolumeLeaders(int num) throws SB_Exception;
    public Vector getNYSETopVolumeLeaders(int num) throws SB_Exception;
    public Vector getAMEXTopVolumeLeaders(int num) throws SB_Exception;

    public Vector getNASDAQTopPercentageLeaders(int num) throws SB_Exception;
    public Vector getNYSETopPercentageLeaders(int num) throws SB_Exception;
    public Vector getAMEXTopPercentageLeaders(int num) throws SB_Exception;

    public Vector getHistoricalData(String symbol, Calendar startDay, Calendar
endDay) throws SB_Exception;
    public boolean checkIfMarketsOpen() throws SB_Exception;
}

```

Fig.128

```

/**
 * Provides access to the thread wait and notify methods. This is used when an
 * object that is not the thread owner is running and wants wait/notify control
 * over its thread.
 */
public interface RemoteThreadMonitor {

    /**
     * Remote equivalent of Object.wait() .
     */
    public void remoteWait();

    /**
     * Remote equivalent of Object.notifyAll() .
     */
    public void remoteNotifyAll();
}

```

Fig./2c